Universidad de San Carlos de Guatemala
Facultad de ingeniería
Escuela de mecánica eléctrica
Laboratorio de electrónica 6
Segundo semestre de 2015

Microcontrolador básico sintetizado

Objetivos

General

Aplicar los conceptos obtenidos en el laboratorio de electrónica 6 para describir hardware dentro de una FPGA.

Específicos

- ✓ Aplicar los conocimientos adquiridos en cursos anteriores para describir una arquitectura de hardware.
- ✓ Sintetizar los módulos de un microprocesador dentro de una FPGA.
- ✓ Desarrollar la lógica de los procesos concurrentes y secuenciales en el lenguaje de descripción de hardware VHDL.
- ✓ Implementar el manejo del puntero de instrucciones en la pila.
- ✓ Emplear un intérprete básico para programar el hardware descrito.

Descripción

Se debe sintetizar un hardware capaz de funcionar como un microcontrolador muy básico. Este está compuesto por registros, memorias y unidades de funcionamiento lógico. El microcontrolador deberá ejecutar instrucciones, las cuales están almacenadas en una memoria interna. Este al estar procesando cada instrucción, dependiendo de cuál sea, se deberá modificar registros, puertos, espacios de memoria, etc. El manejo del puntero será muy importante ya que este es el encargado de ir ejecutando cada una de las instrucciones, tomar en cuenta los saltos y las llamadas a subrutinas. Todas las instrucciones serán programadas por un usuario, por lo tanto, deberá crear un software capaz de interpretar cada una de las instrucciones creadas en lenguaje assembler y convertirlas a binario. Las instrucciones binarias serán enviadas por el protocolo de comunicación UART, desde el puerto serial de una computadora hasta el puerto serial de una FPGA. Cada instrucción binaria tendrá un formato específico para luego ser decodificada e interpretada por el microcontrolador sintetizado.

Especificaciones

Formato de instrucción binaria

Para enviar por comunicación UART se deberán presentar los siguientes formatos posibles.

Esto depende de qué instrucción se quiera interpretar. El tamaño de la trama de bits no se debe modificar. La longitud total de la trama a enviar por UART es de 22 bits.

Neumónico	Dirección	Dirección
[5 bits]	[8 bits]	[9 bits]

Neumónico	Dirección	Dato
[5 bits]	[8 bits]	[9 bits]

Neumónico	Indicador	Dirección
[5 bits]	[8 bits]	[9 bits]

Neumónico	Dirección
[5 bits]	[9 bits]

Neumónico
[5 bits]

Registros requeridos

Estos registros fueron tomados del microprocesador Z80 por su simplicidad, se puede referir al manual de instrucciones de este microprocesador:

- A: Este es un registro de 8 bits que funciona como el acumulador. Este registro se utiliza generalmente como destino de muchas operaciones aritméticas, de comparaciones y testeos.

- B, C, D, E, H, L: Estos son registros de 8 bits para propósito general, utilizados para operaciones, almacenamiento de valores, etc.

- SP: Registro de 16 bits llamado puntero de pila, este apunta a la posición actual del lector de la pila.

- PC: El *Program Counter* o Contador de Programa es un registro de 16 bits. El cual contiene la dirección de la instrucción actual a ejecutar. El PC no se modifica directamente moviendo valores a este registro, solamente se modificará mediante instrucciones de salto (JP, JR, CALL, etc.).

- Registro de flags: Será llamado F, este no es un registro de propósito general en donde se pueda introducir valores a voluntad. Cada uno de los diferentes bits del registro F tiene un significado propio que cambia automáticamente según el resultado de las operaciones anteriores.

  Los bits son los siguientes:

  - Flag S (sign o signo): Este flag se pone a uno si el resultado de la operación realizada en complemento a dos es negativo (es una copia del bit más significativo del resultado).

  - Flag Z (zero o cero): Este flag se pone a uno si el resultado de la última operación que afecte a los flags es cero.

  - Flag P/V (Parity/Overflow o Paridad/Desbordamiento): En las operaciones que modifican el bit de paridad, este bit vale 1 si el número de unos del resultado de la operación es par, y 0 si es impar. Si, por contra, el resultado de la operación realizada necesita más bits para ser representado de los que nos provee el registro, tendremos un desbordamiento, con este flag a 1. Este mismo bit sirve pues para 2 tareas, y nos indicará una u otra (paridad o desbordamiento) según sea el tipo de operación que hayamos realizado. El flag de desbordamiento se activará cuando en determinadas operaciones pasemos de valores 11111111b a 00000000b, por “falta de bits” para representar el resultado o viceversa.

  - Flag N (Substract o Resta): Se pone a 1 si la última operación realizada fue una resta. Se utiliza en operaciones aritméticas.

  - Flag C (Carry o Acarreo): Este flag se pone a uno si el resultado de la operación anterior no cupo en el registro y necesita un bit extra para ser representado. Este bit es ese bit extra. Veremos su uso cuando tratemos las operaciones aritméticas, en esta misma entrega.

<u>Set de instrucciones</u>

La representación binaria de la instrucción se indica entre corchetes. Son 15 instrucciones básicas a realizar.

- LD [00000]: Instrucción para la carga de datos en los diferentes registros, este no afectará ningún valor del registro de banderas.

  La sintaxis de esta instrucción es:

```
LD DESTINO, ORIGEN
```

  Sus funciones deberán ser las siguientes:

  - Colocar un valor en un registro.

```
LD A, 10        ; A = 10
LD B, 200       ; B = 200
```

  - Copiar el valor de un registro a otro registro.

```
LD A, B         ; A = B
```

  - Escribir en memoria (en una dirección determinada) un valor.

```
LD (12345), 10    ; Memoria[12345] = valor en A
```

  - Escribir en memoria (en una dirección determinada) el contenido de un registro.

```
LD (12345), A     ; Memoria[12345] = valor en A
```

- Asignarle a un registro el contenido de una dirección de memoria.

```
LD A, (12345)    ; A = valor en Memoria[12345]
```

- INC [00001] y DEC [00010]: Estás instrucciones son utilizadas para incremental o decrementar respectivamente el valor de un registro o posición de memoria.

```
LD A, 0      ; A = 0
INC A        ; A = A+1 = 1
LD B, A      ; B = A = 1
INC B        ; B = B+1 = 2
INC B        ; B = B+1 = 3
INC (12345)  ; (12345) = (12345)+1
```

Estas instrucciones activan las banderas de P/V (Paridad y *overflow*) y la bandera C (*Substract* o Resta).

Instrucción	S	Z	P/V	N	C
INC r	*	*	V	0	-
DEC r	*	*	V	1	-

Donde r es un registro de 8 bits, * el flag se ve afectado por la operación acorde al resultado, V cambia cuando existe un *overflow* acorde al resultado y - significa que la instrucción no afecta el resultado.

- ADD [00011]: Instrucción utilizada para realizar sumas
  Sintaxis de la instrucción:

```
ADD DESTINO, ORIGEN
```

Las operaciones requeridas son las siguientes:

```
ADD A, B       ; A = A + B
ADD A, 100     ; A = A + 100
ADD A, (1023)  ; A = A + (1023)
```

Las banderas que se ven afectadas a esta instrucción son:

Instrucción	S	Z	P/V	N	C
ADD	*	*	V	0	*

Donde r es un registro de 8 bits, * el flag se ve afectado por la operación acorde al resultado, V cambia cuando existe un *overflow* acorde al resultado y - significa que la instrucción no afecta el resultado.

- SUB [00100]: Instrucción utilizada para realizar restas.
  Sintaxis de la instrucción:

```
SUB DESTINO, ORIGEN
```

Las operaciones requeridas son las siguientes:

```
SUB A, B        ; A = A - B
SUB A, 100      ; A = A - 100
SUB A, (1023)   ; A = A - (1023)
```

Las banderas que se ven afectadas a esta instrucción son:

Instrucción	S	Z	P/V	N	C
SUB	*	*	V	1	*

Donde r es un registro de 8 bits, * el flag se ve afectado por la operación acorde al resultado, V cambia cuando existe un *overflow* acorde al resultado y - significa que la instrucción no afecta el resultado.

- NOP [00101]: Esta instrucción dejará pasar la línea de instrucción sin hacer nada.

- JP [00110]: Instrucción de salto, realiza un traslado a una sección específica de la pila de instrucciones.

Se deberá implementar dos tipos de saltos:

➔ Salto para un bucle infinito:

```
bucle:
        LD A, 20
        NOP
        (...)
        JP bucle
```

➔ Salto para generar un bucle condicional:

```
bucle:
      DEC D
      JP N , bucle
```

La instrucción de salto condicional deberá funcionar con los siguientes estados del registro de flags F:

- ✓ **JP NZ, direccion**: Salta si el indicador de cero (Z) está a cero (resultado no cero).
- ✓ **JP Z, direccion**: Salta si el indicador de cero (Z) está a uno (resultado cero).
- ✓ **JP NC, direccion**: Salta si el indicador de carry (C) está a cero.
- ✓ **JP C, direccion**: Salta si el indicador de carry (C) está a uno.
- ✓ **JP PO, direccion**: Salta si el indicador de paridad/desbordamiento (P/V) está a cero.
- ✓ **JP PE, direccion**: Salta si el indicador de paridad/desbordamiento (P/V) está a uno.
- ✓ **JP P, direccion**: Salta si el indicador de signo S está a cero (resultado positivo).
- ✓ **JP M, direccion**: Salta si el indicador de signo S está a uno (resultado negativo).

❖ CALL [00111]: Esta instrucción genera una subrutina y retornará la instrucción siguiente en donde se hizo la llamada cuando se encuentre la instrucción RET.

```
LLAMADA:
LD A, 20
(...)
RET
(...)
CALL LLAMADA
```

Recordar que la pila de llamadas es limitada ya que cada posición de salto se guarda en una memoria tipo LIFO (*last in first out*). Esta memoria está contenida en la memoria donde se encuentran los registros. El número de llamadas máximo será de 4.

- AND [01000]: Instrucción que realiza la operación lógica and.
  Las banderas que se ven afectadas a esta instrucción son:

Instrucción	S	Z	P/V	N	C
SUB	*	*	V	0	0

Donde * significa que el flag se ve afectado por la operación acorde al resultado, V cambia cuando existe un *overflow* acorde al resultado y -significa que la instrucción no afecta el resultado.

- OR [01001]: Instrucción que realiza la operación lógica or.
  Las banderas que se ven afectadas a esta instrucción son:

Instrucción	S	Z	P/V	N	C
SUB	*	*	V	0	0

Donde * significa que el flag se ve afectado por la operación acorde al resultado, V cambia cuando existe un *overflow* acorde al resultado y -significa que la instrucción no afecta el resultado.

- XOR [01010]: Instrucción que realiza la operación lógica xor.
  Las banderas que se ven afectadas a esta instrucción son:

Instrucción	S	Z	P/V	N	C
SUB	*	*	V	0	0

Donde * significa que el flag se ve afectado por la operación acorde al resultado, V cambia cuando existe un *overflow* acorde al resultado y -significa que la instrucción no afecta el resultado.

- NEG [01011]: Instrucción que realiza la operación lógica not. Las banderas que se ven afectadas a esta instrucción son:

Instrucción	S	Z	P/V	N	C
SUB	-	-	-	0	0

Donde r es un registro de 8 bits, * el flag se ve afectado por la operación acorde al resultado, V cambia cuando existe un *overflow* acorde al resultado y -significa que la instrucción no afecta el resultado.

- HALT [01100]: La instrucción HALT suspende el funcionamiento de la CPU hasta que una interrupción la habilita o se recibe un reset. Mientras existe el estado HALT, el procesador ejecuta NOP para mantener la lógica de la memoria.

- IN [01110]: Esta instrucción se utilizará para acceder a una dirección específica de puerto para ser guardado en un registro.

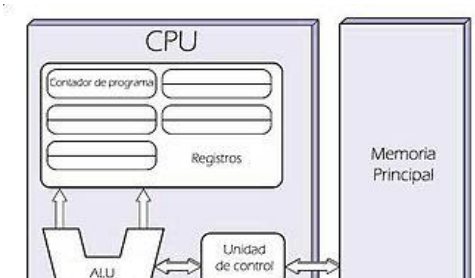
```
; El valor que se encuentra en la dirección de memoria n se guarda en el r
egistro A
IN A, (n)
```

- OUT [01111]: Esta instrucción se utilizará para mostrar en una dirección específica de puerto el registro especificado.

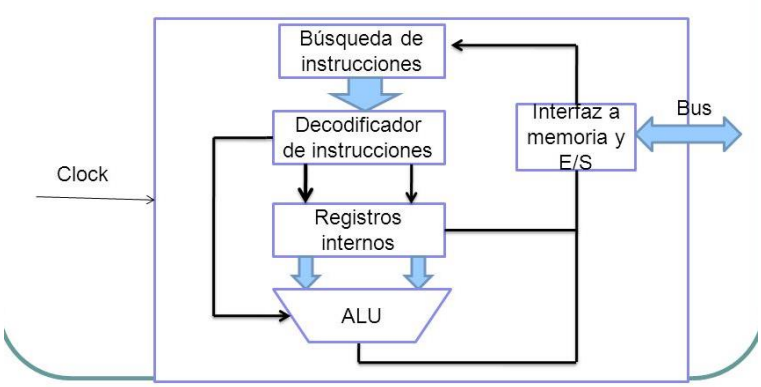
```
; El valor que se encuentra en la dirección de memoria n se guarda en el reg
istro A
OUT (n), A
```

<u>Especificación del Hardware</u>

El dispositivo debe constar con una unidad central de procesos, constituida por memorias para el manejo de los registros y pilas descritos anteriormente, una unidad de control y una unidad lógica aritmética, la cual podrá manejar entradas y salidas de espacios de memoria, como banderas, espacios en memoria RAM y puertos. La unidad de control puede acceder a los registros, a la memoria y a los puertos conectados. Las memorias a implementar son dos, una memoria ROM en la cual se guardará cada una de las instrucciones a ejecutar y una memoria RAM en la cual se guardarán los registros, las *flags*, la LIFO para manejar las llamadas y los puertos de entrada y salida. Se requiere de dos puertos, estos pueden ser de entrada o de salida. El siguiente diagrama muestra un esquema del hardware posible a describir.



La lógica del comportamiento del hardware se muestra en el siguiente diagrama de bloques básico de un procesador.



Las instrucciones deberán ser recibidas a la FPGA por medio de comunicación UART, estas deberán ser guardadas en una memoria ROM interna para ser leídas al ejecutar el programa recibido. Deberán llevar en el registro específico de qué instrucción se está ejecutando. El periodo de reloj deberá ser modificable a uno más largo al que utiliza la FPGA y con esto verificar si cada instrucción se está ejecutando correctamente.

Especificación del software intérprete

El intérprete simplemente deberá leer un archivo, línea por línea guardarlo en arreglos de *Strings* y separarlo en partes las cuales se convertirán en binario, dependiendo de cada instrucción. Si el neumónico está seguido por otra palabra, estos deberán estar separados por un espacio en blanco. Las direcciones y los datos tendrán que estar separados por comas. Si se desea acceder a una posición de memoria este debe estar limitado por paréntesis. Al enviar los datos solamente se trabajará con valores en sistema decimal enteros positivos o negativos, utilizar complemento a dos.

Asumir que el código no tiene errores, no se pide que se detecten errores solamente que se interpreten todas las instrucciones. Al momento de la calificación se proporcionará un código de assembler para que sea programado en el

microcontrolador sintetizado y así poder calificar cada una de las partes que se requieren. Este intérprete puede programarse en Python, Java o Matlab-Octave, creando también la comunicación serial entre la computadora y la FPGA.

Libertades y restricciones

- El hardware se debe sintetizar dentro de una FPGA.
- El número de neumónicos total será de 15.
- No se permite el uso de microcontroladores o microprocesadores que no sean sintetizados dentro de la FPGA.
- El software para sintetizar deberá ser creado en un lenguaje de programación, como Python, java, Matlab, etc.
- Para programar el microcontrolador se deberá utilizar comunicación UART**.**
- **Debe entregar un reporte impreso con el formato de la IEEE que contenga el diagrama esquemático de hardware y el código del intérprete al momento de presentar la práctica.**
- Máximo, 3 integrantes por grupo de trabajo.
- No será necesario utilizar placas externas, por lo tanto deberá estar solamente la tarjeta de la FPGA. Si se desea utilizar el módulo serial-TTL, solamente estarán los cables a las entradas de la FPGA sin placas.
- Solo los proyectos que estén totalmente funcionales, bien encapsulados y que cumplan con todos los requerimientos y restricciones del proyecto pueden ir a Expolab.
- **Si se detecta que dos o más grupos tienen un código de programación similar o totalmente copiado, la nota del proyecto de estos grupos quedara anulada.**
- **La nota total del proyecto del estudiante será anulada, si no tiene conocimiento del hardware sintetizado o de la programación del proyecto.**

Vo. Bo.: Ing. Byron Arrivillaga
Coordinador de Laboratorios